Министерство образования республики Беларусь

Учреждение образования

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. П. О. Сухого

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Отчет по лабораторной работе №1

по дисциплине «Избранные главы информатики»

на тему: «Разработка модели предметной области с использованием ENTITY FRAMEWORK»

Выполнил студент группы ИП-31:

Михалёв А.А.

Принял преподаватель:

Асенчик О.Д.

Гомель 2018

**Цель работы:** Ознакомиться с возможностями ENTITY FRAMEWORK для моделирования предметной области, выборки, вставки, удаления и обновления данных объектов, связанных с таблицами базы данных заданной предметной области.

**Задание:**

1. Создать с использованием .NET Core Entity Framework Core консольное приложение, содержащее набор классов, моделирующих предметную область, и осуществляющих генерацию и заполнение тестовыми наборами записей базу данных MS SQL Server или Sqlite. Для этого необходимо создать:

· Классы моделирующие не менее чем три таблицы базы данных согласно вашему варианту. Перечень таблиц предварительно согласовывается с преподавателем.

Одна из таблиц обязательно должна находиться на стороне отношения «многие» связи с другой таблицей в схеме базы данных.

· Класс контекста данных.

2. Выполнить, используя объекты Entity Framework и LINQ:

1. Выборку всех данных из таблицы, стоящей в схеме базы данных нас стороне отношения «один» – 1 шт.

2. Выборку данных из таблицы, стоящей в схеме базы данных нас стороне отношения «один», отфильтрованные по определенному условию, налагающему ограничения на одно или несколько полей – 1 шт.

3. Выборку данных, сгруппированных по любому из полей данных с выводом какого-либо итогового результата (min, max, avg, сount или др.) по выбранному полю из таблицы, стоящей в схеме базы данных нас стороне отношения «многие» – 1 шт.

4. Выборку данных из двух полей двух таблиц, связанных между собой отношением «один-ко-многим» – 1 шт.

5. Выборку данных из двух таблиц, связанных между собой отношением «один-ко-многим» и отфильтрованным по некоторому условию, налагающему ограничения на значения одного или нескольких полей – 1 шт.

6. Вставку данных в таблицы, стоящей на стороне отношения «Один» – 1 шт.

7. Вставку данных в таблицы, стоящей на стороне отношения «Многие» – 1 шт.:

8. Удаление данных из таблицы, стоящей на стороне отношения «Один» – 1 шт.

9. Удаление данных из таблицы, стоящей на стороне отношения «Многие» – 1 шт.

10. Обновление удовлетворяющих определенному условию записей в любой из таблиц базы данных – 1 шт.

Результаты выборки выводить на экран.

**Листинг программы:**

**Модели:**

**Client.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace lab1.Models

{

public class Client

{

public Client()

{

Tours = new List<Tour>();

}

public int Id { get; set; }

public string Name { get; set; }

public DateTime Birthday { get; set; }

public string Phone { get; set; }

public virtual ICollection<Tour> Tours { get; set; }

public override string ToString()

{

return "{ Id = "+Id +

", Name = " + Name +

", Birthday = " + Birthday +

", Phone = " + Phone + " }";

}

}

}

**Tour.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace lab1.Models

{

public class Tour

{

public int Id { get; set; }

public double Price { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public int TourKindId { get; set; }

public int ClientId { get; set; }

public virtual TourKind TourKind { get; set; }

public virtual Client Client{ get; set; }

public override string ToString()

{

return "{ Id = " + Id +

", Price = " + Price +

", StartDate = " + StartDate +

", EndDate = " + EndDate +

", TourKindId = " + TourKindId +

", ClientId = " + ClientId + " }";

}

}

}

**TourKind.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace lab1.Models

{

public class TourKind

{

public TourKind()

{

Tours = new List<Tour>();

}

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public string Constraints { get; set; }

public virtual ICollection<Tour> Tours { get; set; }

public override string ToString()

{

return "{ Id = " + Id +

", Name = " + Name +

", Description = " + Description +

", Constraints = " + Constraints + " }";

}

}

}

**Контекст данных:**

using lab1.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using System.IO;

namespace lab1.Data

{

public class ToursDbContext : DbContext

{

public DbSet<Client> Clients { get; set; }

public DbSet<Tour> Tours { get; set; }

public DbSet<TourKind> TourKinds { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var builder = new ConfigurationBuilder();

// set path to current directory

builder.SetBasePath(Directory.GetCurrentDirectory());

// get configuration from file appsettings.json

builder.AddJsonFile("appsettings.json");

// create configuration

var config = builder.Build();

string connectionString = config.GetConnectionString("SQLConnection");

var options = optionsBuilder

.UseSqlServer(connectionString)

.Options;

}

}

}

**Строка подключения к базе данных:**

{

"ConnectionStrings": {

"SQLConnection": "Server=(localdb)\\MSSqlLocalDb;Database=ToursAgency;Trusted\_Connection=True;"

}

}

**Исходные коды методов, реализующих требуемые по заданию LINQ запросы:**

using lab1.Data;

using lab1.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace lab1.Utils

{

class LinqHandler

{

private const int selectCount=5;

private ToursDbContext \_db;

private static LinqHandler instance;

public LinqHandler()

{

\_db = new ToursDbContext();

DbInitializer.Initialize(\_db);

}

public static LinqHandler GetInstance()

{

if (instance == null)

instance = new LinqHandler();

return instance;

}

public ToursDbContext DbContext { get { return \_db; } }

public static void Print(string caption, IEnumerable items)

{

Console.WriteLine(caption);

foreach (var item in items)

{

Console.WriteLine(item.ToString());

}

Console.WriteLine();

}

public ICollection SelectTourKinds()

{

var tourKinds = \_db.TourKinds

.OrderBy(tourKind => tourKind.Name);

return tourKinds.Take(selectCount).ToList();

}

public ICollection SelectClients()

{

var clients = \_db.Clients

.OrderBy(client => client.Name);

return clients.Take(selectCount).ToList();

}

public ICollection SelectClientsByName(string name)

{

var clients = \_db.Clients

.Where(client => client.Name.Contains(name))

.OrderBy(client => client.Name);

return clients.Take(selectCount).ToList();

}

public ICollection SelectTourkindSumByName()

{

var tours = \_db.Tours

.Include(t => t.TourKind)

.Include(t => t.Client)

.GroupBy(t => t.TourKind.Name, t => t.Price)

.Select(gr => new

{

TourKindName = gr.Key,

Summ = gr.Sum()

})

.OrderBy(t => t.TourKindName);

return tours.Take(selectCount).ToList();

}

public ICollection SelectJoinedTours()

{

var tours = \_db.Tours

.Include(t => t.TourKind)

.Include(t => t.Client)

.Select(tour => new

{

Id = tour.Id,

Price = tour.Price,

StartDate = tour.StartDate,

EndDate = tour.EndDate,

TourKind = tour.TourKind.Name,

Client = tour.Client.Name,

})

.OrderBy(t => t.Price);

return tours.Take(selectCount).ToList();

}

public ICollection SelectToursByKindName(string name)

{

var tours = \_db.Tours

.Include(t => t.TourKind)

.Include(t => t.Client)

.Where(tour => tour.TourKind.Name.Contains(name))

.OrderBy(tour => tour.Price);

return tours.Take(selectCount).ToList();

}

public void InsertTourKind(TourKind tourKind)

{

\_db.TourKinds.Add(tourKind);

\_db.SaveChanges();

}

public void InsertTour(Tour tour)

{

\_db.Clients.Add(tour.Client);

\_db.TourKinds.Add(tour.TourKind);

\_db.SaveChanges();

tour.TourKindId = tour.TourKind.Id;

tour.ClientId = tour.Client.Id;

\_db.Tours.Add(tour);

\_db.SaveChanges();

}

public void DeleteTourKindByName(string tourKindName)

{

var tourKinds = \_db.TourKinds

.Where(tourKind => tourKind.Name.Equals(tourKindName));

var tours = \_db.Tours

.Include(tour => tour.TourKind)

.Where(tour => tour.TourKind.Name.Equals(tourKindName));

\_db.Tours.RemoveRange(tours);

\_db.SaveChanges();

\_db.TourKinds.RemoveRange(tourKinds);

\_db.SaveChanges();

}

public void DeleteTourById(int tourId)

{

var tours = \_db.Tours

.Where(tour => tour.Id == tourId);

\_db.Tours.RemoveRange(tours);

\_db.SaveChanges();

}

public void UpdateClient(Client newClient)

{

var oldClient = \_db.Clients

.Where(c => c.Id == newClient.Id)

.FirstOrDefault();

if (oldClient != null)

{

oldClient.Name = newClient.Name;

oldClient.Birthday = newClient.Birthday;

oldClient.Phone = newClient.Phone;

};

\_db.SaveChanges();

}

}

}

**Результаты выполнения:**

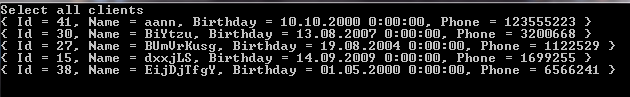


Рисунок 1 – Выборка всех данных из таблицы «Clients»



Рисунок 2 – Выборка данных из таблицы «Clients», отфильтрованные по определенному условию, налагающему ограничения на одно или несколько полей

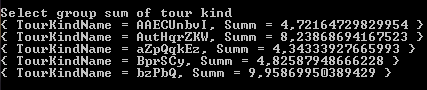


Рисунок 3 – Выборка данных, сгруппированных по полю «tourKindName» с выводом суммы

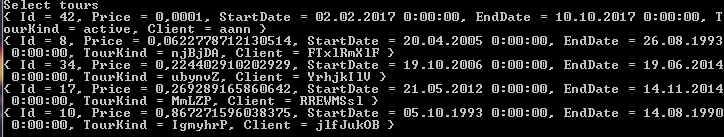


Рисунок 4 – Выборка данных из двух полей таблиц «Tours» и «Clients»

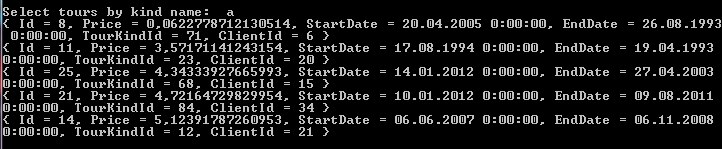


Рисунок 5 – Выборка данных из двух таблиц, связанных между собой отношением «один-ко-многим» и отфильтрованным по некоторому условию



Рисунок 6 – Вставка данных в таблицу «TourKind»



Рисунок 7 – Вставка данных в таблицу «Tours»



Рисунок 8 – Удаление данных из таблицы «TourKind»



Рисунок 9 – Удаление данных из таблицы «Tours»

https://pp.userapi.com/c824700/v824700145/afc8e/ebIs6llzO_A.jpg

Рисунок 10 – Обновление удовлетворяющих определенному условию (id) записей в таблице «Clients»

**Вывод:** в данной лабораторной работе были изучены возможности ENTITY FRAMEWORK для моделирования предметной области, выборки, вставки, удаления и обновления данных объектов, связанных с таблицами базы данных заданной предметной области.